Chapitre-1 : Modèle Relationnel (MR) & Langage SQL – Oracle -

Table des matières

1. Introduction

1.1 Les bases de données

C'est une Collection homogène et structurée de données qui existent sur une longue période de temps et qui décrivent les activités d'une ou plusieurs organisations. C'est aussi un ensemble de données modélisant les objets du monde réel et servant de source de données à des programmes informatique
Cet ensemble structuré **de données** est enregistrées sur des supports informatiques pour satisfaire simultanément plusieurs utilisateurs de façon sélective. Une BDD est gérée par un **Système de Gestion de Base de données (SGBD)**

1.2 Les Systèmes de Gestion de Bases de Données

« DataBase Management Systems » est un Ensemble de logiciels systèmes permettant aux utilisateurs d'insérer, de modifier, et de rechercher efficacement des informations spécifiques dans une grande masse d'informations (pouvant atteindre des Téra octets) partagée par de multiples utilisateurs

Exemples :
MySQL, PostgreSQL (utilisé en TP), Oracle, Microsoft SQLServer ,

Les Principales fonctionnalités d'un SGBD sont :

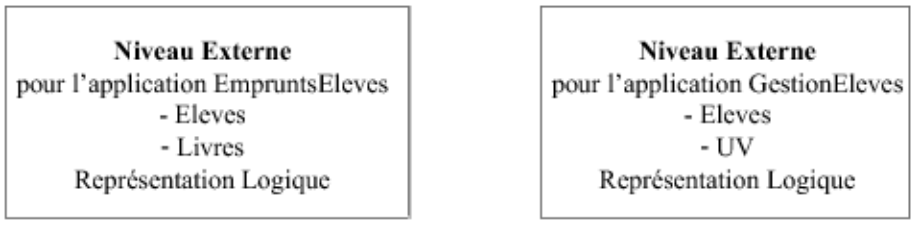
- **Création et mises à jour de la structure de la base de données**. Cette fonctionnalité peut être faite pas le concepteur et/ou le DBA DataBase Administrator.
- **Saisie et mises à jour des données** réalisé par le concepteur et/ou les utilisateurs. ce sont des opérations d'écriture dans les BDD
- **Interrogation des données** selon différents critères et/ou en effectuant des calculs afin d'obtenir des informations. Ces opérations de lecture sont réalisées par les utilisateurs
- **Administration de la base de données en gérant l**es utilisateurs et leurs droits d'accès etc ..Opérations effectués par l' administrateur DBA

1.3 Les niveaux d'abstraction pour les SGBD

L'un des objectifs des SGBD est d'assurer une abstraction des données afin de simplifier l'interaction des utilisateurs avec les bases de données. Les développeurs cachent aux utilisateurs des détails internes non pertinents. Une architecture à trois niveaux d'abstraction est utilisée.

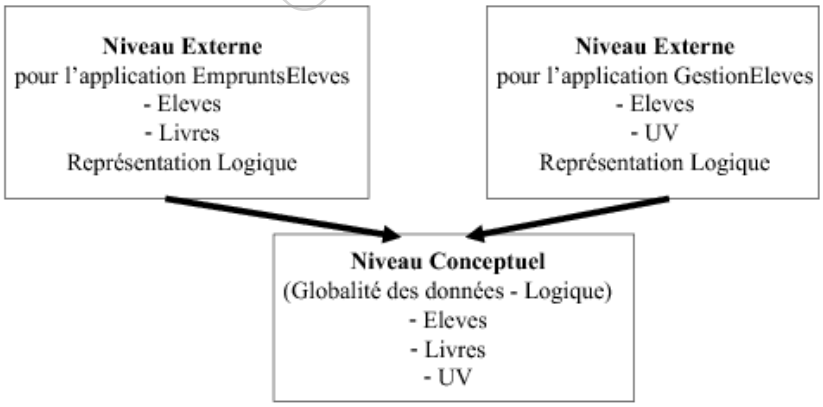
Niveau externe ; c'est niveau qui regroupe les vues utilisateurs. C'est le niveau le plus élevé d'abstraction des données. Il décrit l'interaction de l'utilisateur avec le système de base de données.

Exemple : soit les deux services d'une école : la bibliothèque et la direction.



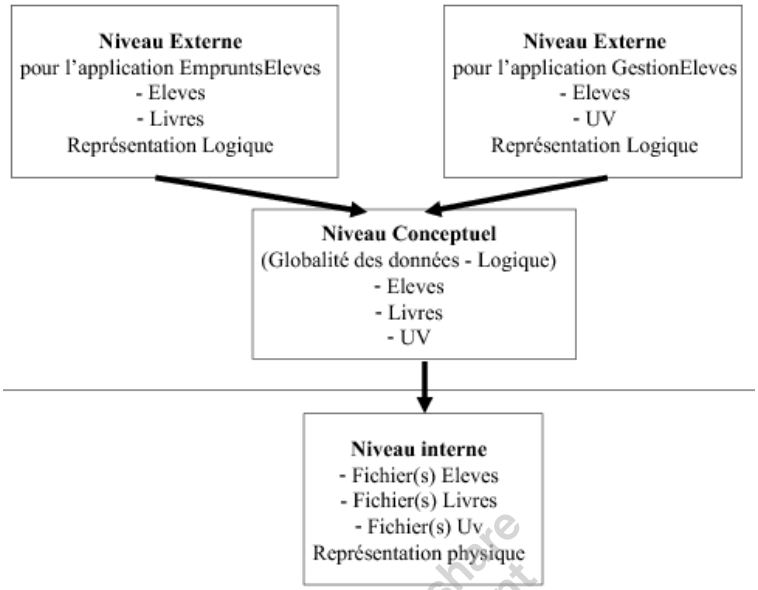
La bibliothèque gère les emprunts des livres faites par les élèves et la direction gère l'inscription des élèves dans les UV. Ces deux services sont indépendants et les personnes qui y travaillent ont besoin des informations sur leurs service et pas sur l'autre service. La bibliothèque n'a pas besoin de connaitre dans quel UV est inscrit l'élève pour lui emprunter un livre. Donc chaque application aura ces propres vues. Mais un problème qui se pose c'est que les deux applications (services) ont des données en communs c'est les élèves. Donc pour bien gérer ce problème ce niveau sera complété par le niveau conceptuel.

Niveau conceptuel : ce niveau décrit la globalité des données qui sont stockées dans la base de données. Dans ce niveau on va trouver les entités élève, Livres et les UV ainsi que les associations c.-à-d. les rapports fonctionnels entre ces entités. Par exemple : l'élève « E » est inscrit dans les UV (UV1, UV2) et a emprunté les livres (L1, L2, L3).



Ces deux premiers niveaux sont des niveaux logiques c.-à-d. sont indépendants de la façon avec lesquelles sont stockées les données dans la mémoire.

Niveau interne : c'est le plus bas niveau. Il décrit comment les données sont réellement stockées dans la base de données. On peut obtenir les détails complexes de la structure des données à ce niveau.



1-4 Les caractéristiques d'un SGBD

L'architecture à trois niveaux permet d'avoir une indépendance entre les données et les traitements. D'une manière générale un SGBD doit avoir les caractéristiques suivantes :

- **Indépendance physique :** le niveau physique peut être modifié indépendamment du niveau conceptuel. Cela signifie que tous les aspects matériels de la base de données n'apparaissent pas pour l'utilisateur, il s'agit simplement d'une structure transparente de représentation des informations
- **Indépendance logique :** le niveau conceptuel doit pouvoir être modifié sans remettre en cause le niveau physique, c'est-à-dire que l'administrateur de la base doit pouvoir la faire évoluer sans que cela gêne les utilisateurs
- **Manipulabilité :** des personnes ne connaissant pas la base de données doivent être capables de décrire leur requête sans faire référence à des éléments techniques de la base de données
- **Rapidité des accès :** le système doit pouvoir fournir les réponses aux requêtes le plus rapidement possible, cela implique des algorithmes de recherche rapides
- **Administration centralisée :** le SGBD doit permettre à l'administrateur de pouvoir manipuler les données, insérer des éléments, vérifier son intégrité de façon centralisée
- **Limitation de la redondance :** le SGBD doit pouvoir éviter dans la mesure du possible des informations redondantes, afin d'éviter d'une part un gaspillage d'espace mémoire mais aussi des erreurs

- **Vérification de l'intégrité :** les données doivent être cohérentes entre elles, de plus lorsque des éléments font référence à d'autres, ces derniers doivent être présents
- **Partageabilité des données :** le SGBD doit permettre l'accès simultané à la base de données par plusieurs utilisateurs
- **Sécurité des données :** le SGBD doit présenter des mécanismes permettant de gérer les droits d'accès aux données selon les utilisateurs

1-5 Instances de base de données

Instances de base de données représente les données à un instant donné. L'évolution des données au cours du temps est réalisée par les trois opérations d'écriture qui sont l'insertion, la mise à jour et la suppression, (insert, update et delete). Ces manipulations sont assurées par un langage de manipulation de données -LMD- (DML-Data Manipulation Language).

1-6 Schéma de base de données

C'est une description de la structure des données. Cet ensemble de définitions est exprimés par un langage de définition de données -LDD- (DDL Data Definition Language

2. Modèle relationnel

2-1 Relation

Ce modèle s'inspire du concept mathématique de **relation**.

Définitions. Soient A_1, A_2, A_3, ..., A_n n ensembles quelconques. On appelle produit cartésien de A_1, A_2, A_3, ..., A_n l'ensemble $A_1 \times A_2 \times A_3 \times ... \times A_n$ défini par :

$$\mathbf{A_1 \times A_2 \times A_3 \times ... \times A_n = \{(x_1, x_2, x_3, ..., x_n) | \ x_1 \in A_1, x_2 \in A_2, x_3 \in A_3, ..., x_n \in A_n\}}$$

On appelle **relation** sur les ensembles A_1, A_2, A_3, ..., A_n un sous - ensemble de leur produit cartésien. A_1, A_2, A_3, ..., A_n s'appelle **domaines**. A chaque relation et à chaque domaine d'une relation correspond un nom.

Exemple. Soient A = {1, 2, 3} et B = {a, b}.
On a : A x B = {(1, a), (2, a), (3, a), (1, b), (2, b), (3, b)}.
Le sous - ensemble $\Re$= {(1, a), (1, b), (2, a), (2, b)} est une relation définie sur les domaines A et B.

La relation $\Re$ peut s'écrire, également, sous la forme :

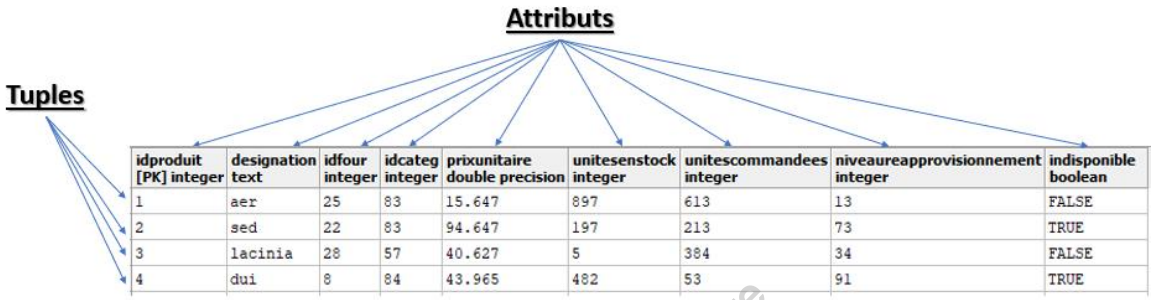
$\Re$	A	B
	1	a
	1	b
	2	a

2	b

Figure-1

Dans le système de notation utilisé ci-dessus (figure 1) la relation $\Re$ est représentée comme un tableau de données. D'où l'appellation de **tableau** ou **table**.
Toute colonne d'une relation s'appelle **attribut** (ou encore **champ** ou **rubrique**). Un attribut est, également, caractérisé par un nom.

Une relation est donc une collection de nuplets (tuples en anglais) décrivant des données de même structure. C'est un tableau composé d'attributs et d'enregistrement.



idproduit [PK] integer	designation text	idfour integer	idcateg integer	prixunitaire double precision	unitesenstock integer	unitescommandees integer	niveaureapprovisionnement integer	indisponible boolean
1	aer	25	83	15.647	897	613	13	FALSE
2	sed	22	83	94.647	197	213	73	TRUE
3	lacinia	28	57	40.627	5	384	34	FALSE
4	dui	8	84	43.965	482	53	91	TRUE

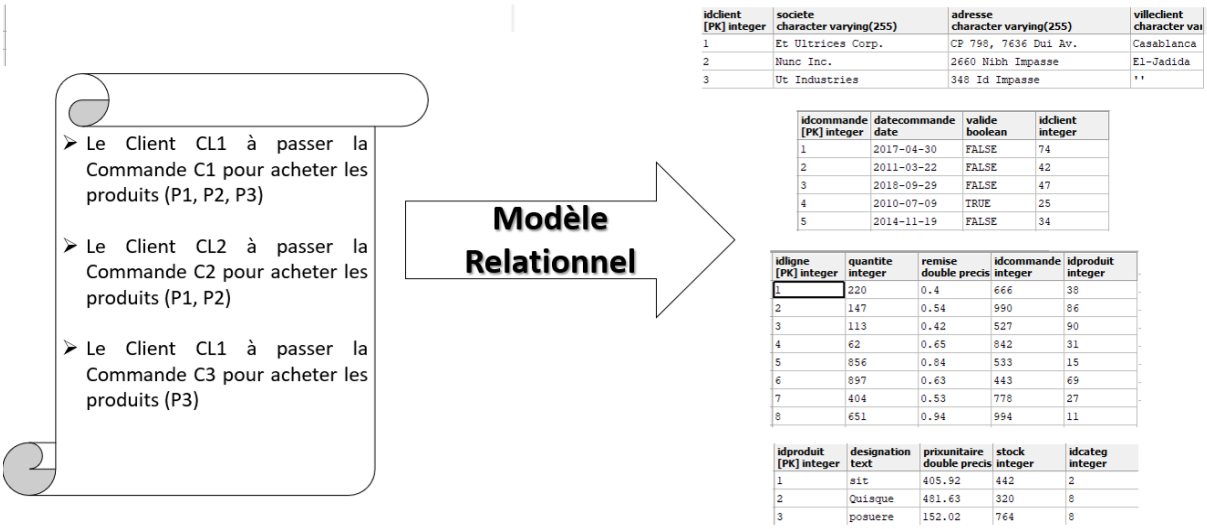
les tuples d'une relation sont tous différents: pas de Tuples en double

Un modèle relationnel est composé d'un ensemble de :

Relation : des sous ensemble du produit cartésien d'une liste de domaines caractérisé par un nom unique représentée sous forme de table composé de colonnes et ensemble de nuplets sans doublon.
Attributs ou champs ou colonnes : ce sont des données qui permettent de décrire une entité (objet) et qui se caractérisent par un nom, un emplacement, un type de données ainsi que d'autres caractéristiques (identifiant par exemple…). Les valeurs prise par un attribut sont atomiques (un attribut à une seule valeur nuplet)
Enregistrements (nuplets) est une donnée composite qui comporte plusieurs champs dans chacun desquels est enregistrée une donnée. Les nuplets d'une relation sont tous différents.

2-2 schéma de relation et schéma de base de données

Le Modèle relationnel permet donc d'organiser les données dans plusieurs relations (tables), ce qui facilitera son exploitation par le SGBD relationnel.



- Schéma de relation est une liste d'attributs et leurs domaines

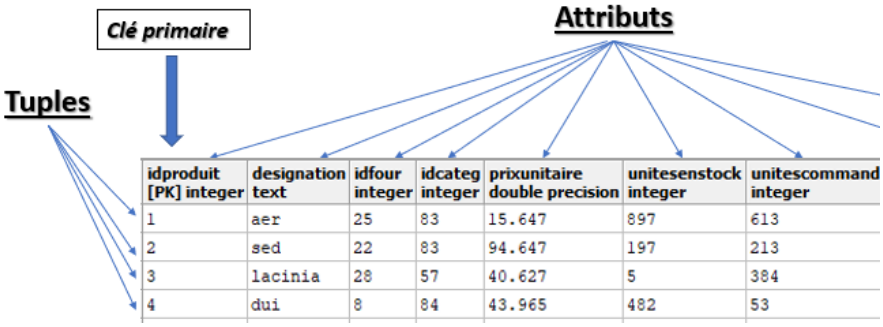
- Un schéma de base de données comprend deux parties :

  - Une description des *tables (schéma de relation)*,
  - Des *contraintes* qui portent sur leur contenu elles permettent d'assurer, *au niveau de la base*, des contrôles sur l'intégrité des données qui s'imposent à toutes les applications accédant à cette base. c.-à-d. conservées les informations qui étaient dissociées dans plusieurs relations.

Un schéma de BDD est défini à l'aide d'un langage de définition des données (LDD) tel que l'SQL.

2-3 Clé primaire

C'est un attribut (ou ensemble d'attributs) permettant d'identifier de manière unique les nuplets de la relation. Les valeurs d'une clé sont uniques (pas de doublon) et non null, c.-à-d. on doit forcément saisir une valeur de la clé pour un tuple donné. Un tuple doit forcement avoir un identifiant.



Un même enregistrement peut très bien avoir plusieurs clés. Dans ce cas, on choisit le plus souvent une clé parmi toutes les autres, qui sera considérée comme une meilleure clé que les autres : c'est la **clé primaire**. Les autres clés seront alors appelées des **clés secondaires** ou clés alternatives

2-4 Clé étrangère (FK Foreign Key)

Attribut (ou ensemble d'attributs) d'une relation qui fait référence à la clé primaire d'une autre relation. Les FK permettent d'éviter les redondances d'information.

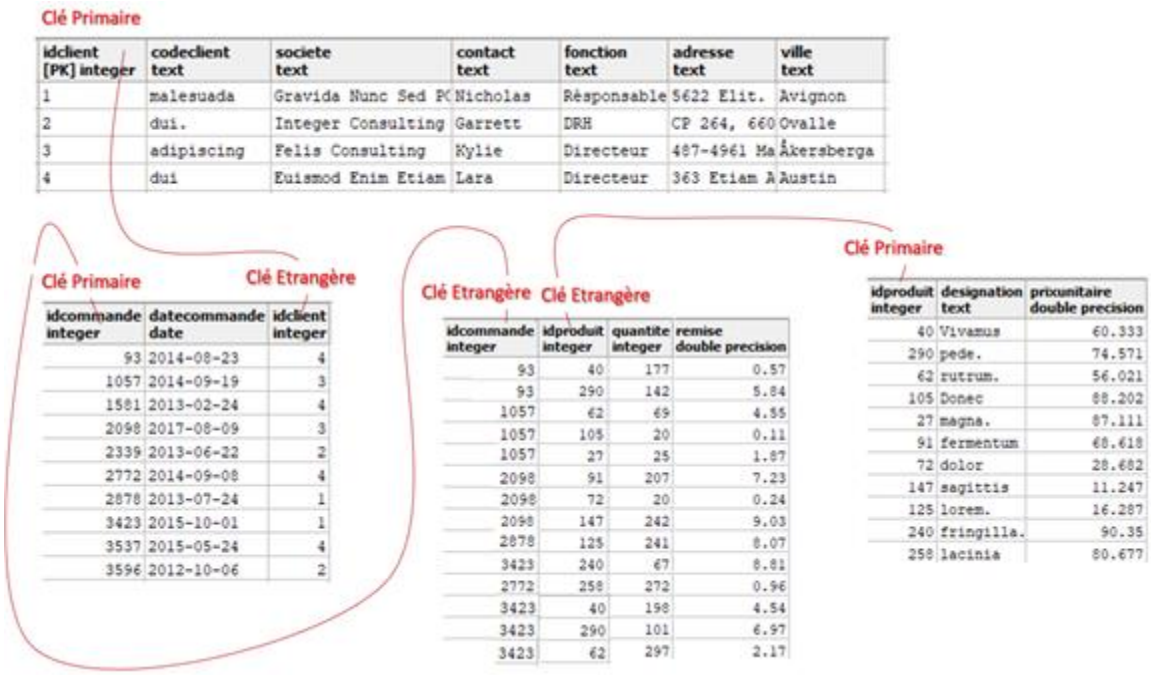
Mauvaise structuration des données : Mettre toutes les données dans une seule relation.

idclient integer	societe text	adresse text	ville text	idcommande integer	datecommande date	idproduit integer	designation text	prixunitaire double precision	quantite integer	remise double precision
1	Gravida Nunc Sed PC	5622 Elit. Avenue	Avignon	4023	2016-10-20	62	rutrum.	56.021	116	5.93
1	Gravida Nunc Sed PC	5622 Elit. Avenue	Avignon	5981	2018-07-27	204	Phasellus	50.787	74	5.22
1	Gravida Nunc Sed PC	5622 Elit. Avenue	Avignon	5484	2019-11-24	72	dolor	28.682	116	8.94
2	Integer Consulting	CP 264, 6605 Volutpa	Ovalle	6942	2012-05-30	79	fames	66.963	141	7.28
2	Integer Consulting	CP 264, 6605 Volutpa	Ovalle	9434	2012-05-19	237	dictum	66.056	120	2.2
2	Integer Consulting	CP 264, 6605 Volutpa	Ovalle	3596	2012-10-06	206	ligula	87.535	12	8.63
2	Integer Consulting	CP 264, 6605 Volutpa	Ovalle	9488	2017-05-14	108	enim	15.622	286	8.19
3	Felis Consulting	487-4961 Magna Impas	Åkersberga	1057	2014-09-19	262	laoreet.	11.511	221	5.65
3	Felis Consulting	487-4961 Magna Impas	Åkersberga	7847	2019-09-28	159	elementum	51.852	50	1.13
4	Euismod Enim Etiam Co	363 Etiam Ave	Austin	6790	2011-03-27	273	adipiscing	14.34	167	3.57
4	Euismod Enim Etiam Co	363 Etiam Ave	Austin	6106	2018-01-10	98	elit.	51.081	224	7.24
4	Euismod Enim Etiam Co	363 Etiam Ave	Austin	6106	2018-01-10	82	venenatis	43.861		

On aura dons ce cas trois anomalies :

- ☞ Anomalies lors de l'insertion :
  - Les données d'un client doivent être saisies autant de fois qu'il y a des commandes.
  - Les données d'un produit doivent être saisies autant de fois qu'il est commandé.
  - Les données d'une commande doivent être saisies autant de fois de produit qui la compose.
- ☞ Anomalies lors d'une modification : à cause de la redondance les mise à jour des données vont être compliquées.
- ☞ Anomalies lors de suppression : Une suppression des données concernant une entité entraîne systématiquement la suppression des données d'une autre entité.

Une bonne présentation est celle qui permet de :
Représenter individuellement chaque entité dans une relation, de manière à ce qu'une action sur l'un n'entraîne pas systématiquement une action sur l'autre.
Définir une méthode *d'identification* de chaque relation, qui permette d'assurer que la même information est représentée une seule fois.
Préserver le lien entre les différentes relations en introduisant dans chaque relation les identifiants des tuples liés à cette relation sans introduire de redondance.



idclient [PK] integer	codeclient text	societe text	contact text	fonction text	adresse text	ville text
1	malesuada	Gravida Nunc Sed P(	Nicholas	Responsable	5622 Elit.	Avignon
2	dui.	Integer Consulting	Garrett	DRH	CP 264, 660	Ovalle
3	adipiscing	Felis Consulting	Kylie	Directeur	487-4961 Ma	Åkersberga
4	dui	Euismod Enim Etiam	Lara	Directeur	363 Etiam A	Austin

idcommande integer	datecommande date	idclient integer
93	2014-08-23	4
1057	2014-09-19	3
1581	2013-02-24	4
2098	2017-08-09	3
2339	2013-06-22	2
2772	2014-09-08	4
2878	2013-07-24	1
3423	2015-10-01	1
3537	2015-05-24	4
3596	2012-10-06	2

idcommande integer	idproduit integer	quantite integer	remise double precision
93	40	177	0.57
93	290	142	5.84
1057	62	69	4.55
1057	105	20	0.11
1057	27	25	1.87
2098	91	207	7.23
2098	72	20	0.24
2098	147	242	9.03
2878	125	241	8.07
3423	240	67	8.81
2772	258	272	0.96
3423	40	198	4.54
3423	290	101	6.97
3423	62	297	2.17

idproduit integer	designation text	prixunitaire double precision
40	Vivamus	60.333
290	pede.	74.571
62	rutrum.	56.021
105	Donec	88.202
27	magna.	87.111
91	fermentum	68.618
72	dolor	28.682
147	sagittis	11.247
125	lorem.	16.287
240	fringilla.	90.35
258	lacinia	80.677

Avec cette présentation on a une saisie unique des informations de chaque client, chaque commande et chaque produit.
L'attribut « idclient » de la relation « Commandes » est une clé étrangère qui fait référence à l'attribut « idclient » de la relation « Clients ».
L'attribut « idcommande » de la relation « LigneCommandes » est une clé étrangère qui fait référence à l'attribut « idcommande » de la relation « Commandes ».
L'attribut « idproduit » de la relation « LigneCommandes » est une clé étrangère qui fait référence à l'attribut « idproduit » de la relation « Produits ».

2-5 Intégrité référentielle

L'Intégrité référentielle est un ensemble de règles garantissant la cohérence des données réparties dans plusieurs relations.

- **Une insertion** dans la relation « Commandes » est autorisée uniquement si on spécifie une valeur « idclient » qui existe dans la relation « Clients ».
  Une insertion dans la relation « LigneCommandes » est autorisée uniquement si on spécifie des valeurs de « idcommande » et « idproduit » qui existent respectivement dans les relations « Commandes » et « Produits ».

- **Une Suppression ou une mise à jour** d'un nuplet de la valeur de clé primaire dans la relation « Clients » est :
  - Non autorisée si un nuplet dans « Commandes » fait référence au nuplet supprimé/mis à jour dans « Clients »
  - Autorisée mais avec suppression/mise à jour en cascade des nuplets correspondant dans « Commandes »
  - Autorisée avec affectation d'une valeur NULL ou d'une valeur par défaut à la clé étrangère des nuplets correspondants dans la relation référençant

2-6 Contraintes d'Intégrités

Ce sont les règles implicites ou explicites qui permettent de garantir la cohérence des données lors des mises à jour de la base. En définissant ces contraintes le SGBD va refuser toutes Ecriture qui viole ces contraintes. Parmi ces contraintes on site :

- **Contraintes d'intégrité référentielle :** Définition des règles qui assure le respect de l'intégrité référentielle

- **Valeur nulle (NULL) :**
  - Valeur conventionnelle introduite dans une relation pour représenter une information inconnue ou inapplicable
  - Tout attribut peut prendre une valeur nulle exceptés les Attributs de la clé primaire (contrainte d'entité)
  - Toute clé étrangère peut prendre une valeur nulle

- **Contraintes d'identité :** tout nuplet doit posséder une valeur de clé primaire unique.

- **Contraintes de domaine :** les valeurs de certains attributs doivent être prises dans un ensemble valeur bien définies.

- **Contraintes d'unicité :** une valeur d'attribut ne peut pas être affectée deux fois.

- **Contraintes générales :** règle permettant de conserver la cohérence de la base de manière générale.

Exemple :

Contraintes de domaine : La ville d'un client prend sa valeur dans l'ensemble {Casablanca, Rabat, Marrakeck, ….}

Contraintes d'unicité : Un client, identifié par son idclient, a un « codeclient » unique (il n'y a pas deux clients qui ont le même code)

Contraintes générales : La date de création d'une commande doit être antérieure à la date de livraison.

2-7 Création d'une base de données.

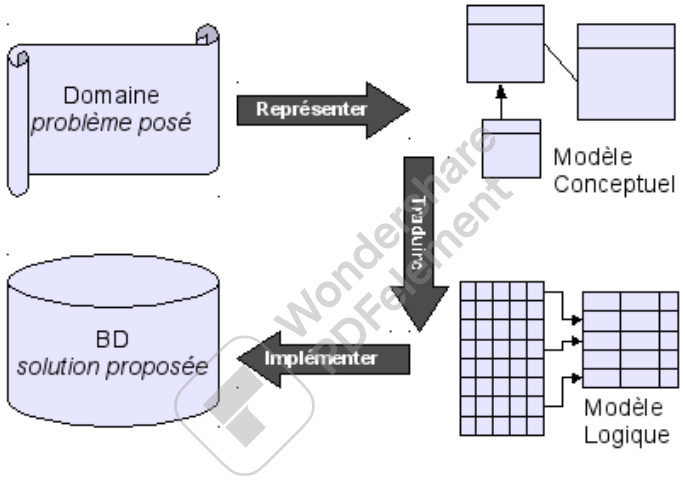
La création d'une base de données passe par trois étapes :

a- **Création d'un modèle conceptuel des données (MCD):** permet de décrire d'une manière simplifié le réel (ensemble de règles de gestion). Cette description se fait d'une façon graphique en utilisant des diagrammes composés des entités et des associations.

b- **Création du modèle Logique des données (MLD) :** A partie du MCD et en appliquant un ensemble de règles de passage on créer le MLD. Ce dernier permet de décrire la structure de données utilisée sans faire référence à un langage de programmation. Il s'agit donc de préciser le type de données utilisées lors des traitements. Si les structures utilisées pour décrire les données dans des tables on parle du modèle Logique de données relationnel.

1. Pour chaque relation Définir les différents attributs et Définir la clé primaire
2. Pour chaque attribut de chaque relation Définir le type et le domaine et Préciser les propriétés (format, etc…)
3. Quand il y a plusieurs relations définir les clés étrangères

c- **Création du modèle Physique des données (MPD) : L'implémentation :** Elle correspond aux choix techniques, en termes de SGBD choisi et à leur mise en œuvre (programmation, optimisation...).



Exemple-1 : On veut créer une base de données stockant des enseignants (avec leur nom, prénom, spécialité et des départements, chaque enseignant appartenant à un et un seul département

Le Modèle relationnel est :

Departement(IdDept , NomDept)
Enseignant(IdEns, Num, Nom, Prénom, #idDept, Specialite)

Types, Propriétés et contraint :

- Table Département :
  - idDept : entier, clé primaire
  - NomDept : Chaine de char, Unique et nom null
- Table Eneignant :
  - idEns : entier, cléprimaire
  - Num ; Unique, non Null
  - Nom : Chaine de char

- Prenom : Chaine de char
- idDep : entier, clé étrangère les valeurs font référence à Dzpartement(IDDept)
- Spécialité ∈ {"$Mathématique$", "$Informatique$","$Physique$", …}

Exemple-2
On veut créer une base de données gérant des énoncés d'examens et des exercices
Quelles sont les différences entre les modèles relationnels

Création d'une base de données

- Modèle relationnel-1 Enoncé est UNIQUE
  Examen(ExamID, Date, Heure)
  Exercice(ExoID, Enoncé)
  Contenu_Exam(#ExamID, #ExoID, position)

- Modèle relationnel-2 Enoncé est UNIQUE
  Examen(ExamID, Date, Heure)
  Exercice(ExoID, Enoncé ,#ExamID, Position)

- Modèle relationnel-3 Enoncé est UNIQUE
  Examen(ExamID, Date, Heure)
  Exercice(ExoID,#ExamID, Enoncé)

Dans le modèle-1 : Dans un Examen on trouve plusieurs Exercices, et un Exercice se trouve dans plusieurs Examens mais dans des positions différentes.

Dans le modèle-2 : Dans un Examen on trouve plusieurs Exercices, et un Exercice ne se trouve que dans un seul examen au plus. Dans ce cas pour chaque exercice on fait correspondre un seul énoncé.

Dans le modèle-3 : Dans un Examen on trouve plusieurs Exercices et un Exercice se trouve dans plusieurs Examens. Un exercice donné à des énoncés différents en fonction de l'examen. Un exercice ne peut pas être identifié par « ExoID » seulement mais par le couple (ExoID, ExamID)

2.8. Les Requêtes

Une requête est une interrogation d'une base de données. Elle peut comporter un certain nombre de critères pour préciser la demande. Il existe plusieurs langages de requêtes, qui sont spécifiques à la structure des bases de données. Le plus connu est le SQL, il est exploité dans les bases de données relationnelles (dont les informations sont enregistrées dans des tableaux à deux dimensions). Il existe trois types de requête

1. Requêtes d'interrogation (Data Manipulation Language)
2. Requêtes d'insertion, de mise à jour et de suppression des données (Data Manipulation Language)
3. Requêtes de définition de schéma Data (Definition Language)

3. Langage des requêtes structurées – SQL -

Les applications sont programmées avec des langages informatiques externe aux serveurs du SGBD (C, C++ PHP, python, Java, etc…) mais ces applications consomment en générale des données structurées dans des bases de données. Pour récupérer ces données on utilise le langage SQL (Structured Query Language). Les instructions SQL s'écrivent d'une manière qui ressemble à celle de phrases ordinaires en anglais. Cette ressemblance vise à faciliter l'apprentissage et la lecture.
L'SQL est donc un langage déclaratif, non procédural et ensembliste utilisé par tous les SGBDR.
Non procédurale : ne permet pas de programmer des procédures et fonctions en utilisant des algorithmes.
Ensembliste : permet de récupérer des données stockées dans plusieurs relations. Donc l'SQL utilise les principes de base de l'algèbre relationnelles.
Déclaratif : permet de décrire le résultat escompté, sans décrire la manière de l'obtenir. Les SGBD sont équipés d'optimiseurs de requêtes - des mécanismes qui déterminent automatiquement la manière optimale d'effectuer les opérations entre les relations (tables). Celle-ci est fondée sur des statistiques récoltées à partir des données contenues dans la base de données (nombre de tuple, nombre de valeurs distinctes dans une colonne, etc..).

Le langage SQL est utilisé principalement de deux manières :

- Technique de **l'SQL embarqué** (embedded SQL) : des instructions en langage SQL sont incorporées dans le code source d'un programme écrit dans un autre langage externe.
- Technique des **procédures stockées** : des fonctions et des procédures écrites dans un langage procédural propre à chaque SGBD (PL/SQL pour Oracle, SQL/PSM pour MySQL ...) sont enregistrées dans la base de données en vue d'être exécutées par le SGBD. Cette technique est aussi utilisée pour les triggers - procédures déclenchées automatiquement sur modification du contenu de la base de données.

L'SQL est un langage complet de gestion de bases de données relationnelles. Il permet de :
- définir les données (Langage de Définition de Données LDD) (ordres **CREATE, ALTER, DROP**),
- manipuler les données (Langage de Manipulation de Données LMD) (ordres **UPDATE, INSERT, DELETE**)
- interroger une base de données (ordre **SELECT**)
- contrôler l'accès aux données (Langage de Contrôle de Données LCD) (ordres **GRANT, REVOKE**).

L'intérêt de SQL est que c'est un langage utilisé par les principaux SGBDR : Oracle, Access, SQL server, MySQL… Chacun de ces SGBDR a cependant sa propre variante du langage. Dans ce chapitre nous présentons un noyau de commandes disponibles sur l'ensemble de ces SGBDR ainsi que les variantes de l'SQL propre à Oracle.

3.1 La définition des données -LDD-

La première série de commandes sert à la création et à la maintenance de la base de données : création des tables des index et des contraintes d'intégrité, modification de la structure d'une table ou suppression d'une table.

SQL dispose pour cela des instructions suivantes :
ALTER TABLE
CREATE TABLE
CREATE INDEX

3-1-1- La commandes CREATE TABLE

La commande CREATE TABLE permet de créer une table dans la base de données courante. Sa syntaxe est la suivante :

CREATE TABLE NomTable (Nomchamp1 type propriété, Nomchamp2 type propriété,..);

Type de données (syntaxe Oracle) :

Type Caractère :

Type	Description
CHAR(*longueur*)	Ce type est une chaine de caractères ayant une longueur constante maximal de 4000 caractères.
VARCHAR2(*longueur*)	Ce type est un une chaine de caractères ayant une longueur variable entre 0 et 4096 caractères.
LONG	données stocke des chaînes de caractères de longueur variable contenant jusqu'à deux gigaoctets, mais avec de nombreuses restrictions

Type Numérique

Type	Description
NOMBRE [(p [, s])]	Nombre ayant une précision p et une échelle s, Une valeur NUMBER nécessite de 1 à 22 octets
FLOAT	Valeur réel qui nécessite de 1 à 22 octets..
INTEGER	Entier NUMBER(38)

Type Date

Type	Description
DATE	Type date La taille est fixée à 7 octets.
TIMESTAMP	Ce type de données contient les champs datetime YEAR, MONTH, DAY, HOUR, MINUTE et SECOND. La taille est de 7 ou 11 octets,

Propriété et contraintes

Syntaxe (syntaxe Oracle)	Description
not null	Valeurs d'un attribut non null
unique	Valeurs unique
primary key (PK)	Clé primaire (unique et non null) c'est un index
foreign key (FK)	Clé étrangère
references	Contrainte d'intégrité référentielle (CIR)
set null	Dans le cas de CIR affectation de valeur null pour les FK sur suppression ou mise à jour de PK
default	Dans le cas de CIR affectation de valeur par defaut pour les FK sur suppression ou mise à jour de PK
CHECK(Expressions Logique)	utiliser pour choisir dans la saisie entre vrai et faux (1 ou 0). Utilisé aussi pour introduire des conditions de validation

a- Les séquences

Type incrémentation automatique : Pour définie ce type de données dans Oracle, on doit passer par trois étapes :

1- Création d'une séquence.
2- Création de la table avec un attribut numérique
3- L'activation de la séquence dans l'insertion des valeurs

La syntaxe de création de séquence est la suivante :

CREATE SEQUENCE [schéma.]nomSéquence
[**INCREMENT BY** entier]
[**START WITH** entier]
[{ **MAXVALUE** entier | **NOMAXVALUE** }]
[{ **MINVALUE** entier | **NOMINVALUE** }]

SYNTAXE	DESCRIPTION
INCREMENT BY	intervalle entre les deux valeurs de la séquence (par défaut 1).
START WITH	première valeur de la séquence à générer.
MAXVALUE	valeur maximale de la séquence.
MINVALUE	valeur minimale de la séquence.

Exemple :

```
create SEQUENCE test
START WITH 1;

create table TB(a number(4,0));

insert into TB(a) values (test.nextval);
```

Exemple : Implémenter le modèle relationnel suivant de gestion des ventes.

Clients(idclient, nomclient, adresse, ville**)**
Commades(idcommande, Datecommande, Datelivraison, valide(oui/non), #idclient**)**
Produits(idproduit, nomproduit, Prix, Stock**)**
LigneCommandes(#idcommande, #idproduit, Quantité, Remise**)**

create table clients (idclient integer primary key, nomclient varchar2(20), adresse varchar2(20), ville varchar2(20));

create table commandes (idcommande integer primary key, datecommande Date, datelivraison Date, CHECK (datelivraison >= datecommande), valide int CHECK (valide in (0,1)) not null, idclient int,
foreign key (idclient) references clients(idclient));

create table produits (idproduit integer primary key, nomproduit varchar(20), prix float not null, stock int not null);

create table lignecommandes (idcommande int, idproduit int, quantité int not null, remise float, primary key (idcommande, idproduit), foreign key (idcommande) references commandes(idcommande) on delete cascade, foreign key (idproduit) references produits(idproduit) on delete cascade);

create table **lignecommandes**(idcommande int, idproduit int, quantité int not null, remise float, primary key (idcommande, idproduit), foreign key (idcommande) references commandes(idcommande) on delete cascade, foreign key (idproduit) references produits(idproduit) on delete cascade);

3-1-2- La commande ALTER TABLE
La commande ALTER TABLE permet de modifier la structure d'une table, sa syntaxe est la suivante

```
ALTER TABLE NomTable Action (spécifications du champ);
```

ALTER TABLE permet trois actions, ces actions sont :
ADD Ajoute un champ a une table
DROP Supprime un champ d'une table
MODIFYModifie les caractéristiques d'un champ

Après l'action, on indique, les spécifications du champ de la même façon que pour la commande CREATE TABLE. On ne peut faire qu'une action à la fois (ajout, suppression ou modification dans la même commande).

Exemple :

Ajout d'un champ :
```
alter table clients add "Date naissance" Date;
```

Suppression d'un champ :
```
alter table clients drop "Date naissance";
```

Remarque :

Dans le cas d'un nom d'un attribut ou d'une table qui est composé de plusieurs mots, il faut utiliser les guillemets.

3.2 Les contraintes

Ce sont les règles de validation que doit vérifier les données saisies dans chaque attribut.

3.2.1- Les contraintes de domaine

Il s'agit de définir l'ensemble des valeurs que peut prendre un attribut. Ces contraintes sont décrites dans la définition d'un attribut, directement après son type et sa longueur.

NOT NULL : on impose que l'attribut possède une valeur
DEFAULT : on spécifie une valeur par défaut dont le type doit correspondre au type de l'attribut
UNIQUE : interdit qu'une colonne contienne deux valeurs identiques.
CHECK(condition) : Cette clause permet de spécifier une contrainte qui doit être vérifiée à tout moment par les tuples de la table.

```
create table clients (idclient int primary key, nomclient varchar2(20), adresse varchar2(20), ville varchar2(20) CHECK (ville IN ('Casa', 'Rabat', 'Fès')));
```

```
create table commandes (idcommande int primary key, datecommande Date, datelivraison Date, CHECK (datelivraison >= datecommande), valide CHECK (valise IN (0,1)) not null, idclient int, foreign key (idclient) references clients(idclient));
```

3.2.2- Les contraintes d'intégrité d'identité

PRIMARY KEY : Elles spécifient la clé primaire d'une table via la clause. Elle doit toujours avoir une valeur déterminée et unique pour la table. Quand une clé primaire est constituée de plusieurs attributs (clé segmentée), la clause PRIMARY KEY est placée avant la définition des attributs, séparée par une virgule.

Exemples :

- create table clients (idclient primary key, ….)

- create table Appartement (…, primary key (NumApp, NumImm);

3.2.3- Les contraintes d'intégrité référentielle

foreign key (nonAttribut) references nom_table_référencée(clé candidate) : permet de définir dans une table une clé étrangère qui fait référence à une clé d'une autre table.

Exemples :

```
create table lignecommandes(idcommande int, idproduit int, quantité int not null,
remise float, primary key (idcommande, idproduit),
foreign key (idcommande) references commandes(idcommande),
foreign key (idproduit) references produits(idproduit);
```

Spécification des actions à effectuer en cas de modification (clause **ON UPDATE**) ou de suppression (clause **ON DELETE**) de valeurs de clés référencées :
CASCADE, SET NULL

Exemple :

```
create table lignecommandes(idcommande int, idproduit int, quantité int not null,
remise float, primary key (idcommande, idproduit),
foreign key (idcommande) references commandes(idcommande) on delete cascade,
foreign key (idproduit) references produits(idproduit) on delete cascade);
```


3.3 La Manipulations des Données -LMD-

Une fois les tables créées, on peut commencer à y insérer ou supprimer des enregistrements, mettre à jour des données. Toutes ces opérations sont des opérations de manipulation des bases de données.
Pour effectuer ces manipulations, SQL dispose de 3 instructions :
INSERT
UPDATE
DELETE

3.3.1. La commande INSERT

La commande INSERT est utilisée pour ajouter des enregistrements ou des parties d'enregistrements dans des tables. Elle est utilisée généralement sous deux formes :

1ère forme

```
INSERT INTO table (champ1, champ2,...)
VALUES ('valeur1','valeur2',...);
```

Si des valeurs doivent être insérées dans tous les champs de l'enregistrement de la table, la liste des noms des champs n'a pas besoin d'être explicitement indiquée dans la commande. Les valeurs des champs à insérer doivent cependant apparaître dans le même ordre que les noms des champs lors de la création de la table, sans oublier un seul champ. Faites attention en utilisant cette syntaxe, si la structure de la table change plus tard, la commande qui était bonne risque de ne plus fonctionner

correctement. Il vaut mieux toujours indiquer explicitement le nom des champs sur lesquels on veut agir.

Exemple : on veut insérer trois nouveaux enregistrements dans les tables Clients, Commandes, Produits et lignecommades.

```
insert into clients (nomclient , adresse , ville)
values ('M2M', 'Rue des Racines 45', 'Casablanca');

insert into commandes (datecommande, datelivraison, valide, idclient)
values ('14/12/2021', '16/12/2021', 'yes', 1);

insert into produits (nomproduit, prix, stock) values ('PC HP',
12543.00, 120);
insert into produits (nomproduit, prix, stock) values ('PC DELL',
10543.00, 100),
insert into produits (nomproduit, prix, stock) values ('Imprimante HP',
3000.00, 55);

insert into lignecommandes values (1, 1, 12, 0.12),
insert into lignecommandes values (1, 2, 2, 0.00),
insert into lignecommandes values (1, 3, 15, 0.10);
```

2ème forme

```
INSERT INTO "table1" ("column1", "column2", ...)
SELECT "column3", "column4", ...
FROM "table2"
```

Dans cette seconde forme, le résultat de la requête va être inséré dans les champs indiqués de la table. Cette méthode est utilisée lorsque plusieurs enregistrements sont ajoutés simultanément. Nous devons maintenant utiliser une instruction SELECT pour spécifier les données à ajouter à la table. Nous sommes donc en train d'insérer des données provenant d'une autre table. Dans les deux cas, les valeurs insérées doivent correspondre au type de données du champ dans lequel l'insertion va être faite, on ne peut pas, par exemple demander l'insertion d'une chaîne de caractères dans un champ de type numérique ou monétaire. Les chaînes de caractères doivent être placées entre apostrophes ('), les champs numériques ou vides (NULL) ne doivent pas être placés entre apostrophes.

3.3.2. La commande UPDATE

La commande UPDATE est utilisée pour changer des valeurs dans des champs d'une table. Sa syntaxe est :

```
UPDATE Nomtable
SET champ1 = nouvelle_valeur1,
    champ2 = nouvelle_valeur2,
    champ3 = nouvelle_valeur3
WHERE condition;
```

La clause SET indique quels champs de la table vont être mis à jour et avec quelles valeurs ils vont l'être. Les champs non spécifiés après la clause SET ne seront pas modifiés.
Par exemple, si nous voulons, dans la table produit, augmenter le prix de 20% d'un produit dont le idproduit est 2, nous taperons :

```
update produits set prix=prix+0.25 where idproduit=2;
```

La commande UPDATE affecte tous les enregistrements qui répondent à la condition donnée dans la clause WHERE. Si la clause WHERE est absente, tous les enregistrements de la table seront affectés.

Par exemple, si nous tapons :

```
UPDATE produits
SET prix_unitaire = 1000;
```

Le prix unitaire de TOUS les produits de la table produit va être modifié.
Tout comme la commande INSERT, la commande UPDATE peut contenir une requête. Dans ce cas la syntaxe est la suivante :

```
UPDATE table
SET   champ1 = nouvelle_valeur1,
champ2 = nouvelle_valeur2,
champ3 = nouvelle_valeur3
WHERE condition
(requête);
```


« requête » est une requête faite avec la commande SELECT.

3.3.3. La commande DELETE

Pour supprimer des enregistrements d'une table. La syntaxe est la suivante :

```
DELETE
FROM table
WHERE condition;
```

On ne peut pas supprimer seulement le contenu de quelques champs des enregistrements. La commande DELETE supprime des enregistrements entiers, c'est pour cela qu'il n'est pas nécessaire d'indiquer ici des noms de champs. La condition spécifiée après WHERE va déterminer quels sont les enregistrements à supprimer.
Par exemple, pour supprimer tous les clients dont la ville est Saint-Quentin :

```
DELETE FROM Clients WHERE ville='Casablanca';
```

Pour supprimer tous les enregistrements d'une table, n'indiquez pas de clause WHERE

3.4 L'interrogation des Données -LID- (La commande SELECT)

La commande SELECT est la commande la plus complexe de SQL. Cette commande va servir à faire des requêtes pour récupérer des données dans les tables. Elle peut être associée à une des commandes de manipulation de tables vues avant.

3.4.1. Requêtes simples – La Projection-

Une requête simple est créée à partir d'une seule table. La seule opération algébrique utilisée est la projection. La syntaxe est la suivante :

SELECT `champ1`, `champ2`, `champ3`, ... FROM table;	$\pi_{champ1,champ2,\dots}(table)$

S'il y a plus d'un champ spécifié après SELECT, les champs doivent être séparés par des virgules. Les champs sont retournés dans l'ordre spécifié après la clause SELECT, et non pas dans l'ordre qu'ils ont été créés dans la table.

Par exemple :

Pour sélectionner les champs "prénom" et "nom" de tous les enregistrements de la table Clients :

```
select idclient, nomclient
from clients;
```

Va renvoyer les tuples les attributs « idclient » et « nomclient ».
Si on veut récupérer tous les champs des enregistrements sélectionnés, la syntaxe est la suivante :

```
SELECT *
FROM clients;
```

Les clauses SELECT et FROM doivent obligatoirement apparaître au début de chaque requête,

3.4.2. Requêtes simples avec critère -La sélection -

Dans ces requêtes l'opération algébrique sélection est utilisée par le SGBD. On indique des critères de sélection avec la clause WHERE :

SELECT * FROM table WHERE conditions ;	$\sigma_{conditions}(table)$

Par exemple, pour sélectionner tous les Clients de la table "Clients" dont le code postal est 75000:

```
SELECT *
FROM Clients
WHERE idclient = 7500;
```

On peut utiliser les opérateurs suivants dans la clause WHERE :

Opérateur	Signification
=	Egal
<> ou !=	Différent
<	Inférieur
>	Supérieur
<=	Inférieur ou égal
>=	Supérieur ou égal
BETWEEN	Entre deux valeurs
LIKE	Comme (contient)
NOT LIKE	Ne contient pas
IS NULL	La case est vide
IS NOT NULL	La case n'est pas vide

Exemple : Pour sélectionner tous les articles dont le prix est supérieur à 100 dh:

```
SELECT *
FROM Clients
WHERE prix_unitaire > 100;
```


Dans le cas où on doit utiliser plusieurs critères combinés ou alternatifs, nous devons utiliser les opérateurs logiques AND et OR.

```
SELECT *
FROM Clients
WHERE Ville = 'Casablanca' OR Ville = 'Rabat'
```

a- Opérateur Classement (ORDER BY)

Pour classer les valeurs d'un champ on utilise l'opérateur ORDER BY

- Classement croissant :

```
SELECT `champ1`, `Champ2`
From table
ORDER BY `Champ2`
```

- Classement décroissant :

```
SELECT `champ1`, `Champ2`
From table
ORDER BY `Champ2` DESC
```

b- Clauses IN et BETWEEN

Pour sélectionner des enregistrements dont la valeur d'un champ peut être comprise dans une liste ou entre deux valeurs, on utilise les clauses IN et BETWEEN.
Par exemple : Pour sélectionner les clients vivant à Saint-Quentin ou Paris :

```
SELECT *
FROM Clients
WHERE `ville` IN ('Saint-Quentin', 'Paris');
```

Ou pour sélectionner les produits dont le prix est compris entre 100 et 1000 F :

```
SELECT *
FROM Produits
WHERE `prix_unitaire` BETWEEN 100 AND 1000;
```

Pour sélectionner les produits dont le prix n'est pas dans cet intervalle :

```
SELECT *
FROM Produits
WHERE `prix_unitaire` NOT BETWEEN 100 AND 1000;
```

De la même façon, NOT IN sélectionne les enregistrements exclus de la liste spécifiée après IN.

select *
from clients
where ville **NOT IN** ('Casa', 'Rabat');

c- La clause LIKE

La clause LIKE permet de faire des recherches approximatives sur le contenu d'un champ. Par exemple, pour sélectionner les clients dont le nom commence par la lettre C :

```
SELECT *
FROM Clients
WHERE societe LIKE 'C%';
```

Le symbole % remplace un ensemble de caractères, pour représenter tous les noms commençant par C, on utilisera 'C%', tous ceux se terminant par C, on utilisera '%C', et tous ceux comportant la lettre C : '%C%'.

3.4.3. Requêtes simples avec calcule sur les dates

Ces requêtes utilisent des fonctions SQL de type date.

a- Les fonctions Dates

La fonction Oracle EXTRACT() récupère un champ tel qu'une année, un mois et un jour à partir d'une valeur de date/heure. La syntaxe est donnée par :

EXTRACT(field FROM source)

L'argument field spécifie le champ à extraire de la valeur date/heure.

- DAY: Renvoie le jour de la date
- MONTH: Renvoie le mois de la date
- YEAR: Renvoie l'année de la Date
- WEEK : Renvoie le jour de la semaine d'une date
- CURRENT_DATE: Renvoie la date actuelle (date du système)

Exemple :

Pour sélectionner les commandes réalisées en 2018, on exécute la requête suivante :

SELECT idcommande, datecommande
FROM commandes
where **extract(year from datecommande)=2018;**

	idcommande integer	datecommande date
1	28	2018-03-28
2	33	2018-08-01
3	34	2018-11-21
4	46	2018-03-06
5	69	2018-02-21
6	80	2018-05-28

Pour sélectionner les commandes réalisées cette année, on exécute la requête suivante :

SELECT idcommande, datecommande
FROM commandes
where **extract(year from datecommande)=extract(year from CURRENT_DATE);**

Pour sélectionner les commandes réalisées ce mois, on exécute la requête suivante :

SELECT idcommande, datecommande
FROM commandes
where **extract(month from datecommande)= extract(month from CURRENT_DATE)**
AND extract(year from datecommande) = extract(year from CURRENT_DATE);

3.4.4. Requête avec création de nouveau champ avec formule

Pour créer une requête avec création de nouveau champ on utilise une formule où on utilise les attribut de la table mentionnée dans « From ». On doit donner un nom pour ce nouvel attribut sinon le SGBD va donner un nom par défaut.

```
SELECT `champ 1`, formule AS `nouveau_champ`
From table ;
```

Exemple : dans la BDD des ventes on ajoute un attribut TVA puis on créer une requête qui calcule le prix avec la TVA.

```
Alter table produits add TVA float;

select idproduit, designation, prixunitaire, prixunitaire*(1+TVA) as prixttc
from produits;
```

	idproduit integer	designation text	prixunitaire double precision	prixttc double precision
1	243	est	84.818	89.0589
2	244	diam	51.074	53.6277
3	245	netus	80.531	84.55755
4	246	penatibus	1.916	2.0118
5	247	rhoncus.	82.144	86.2512
6	248	Fusce	91.951	96.54855
7	249	congue.	36.008	37.8084
8	250	magna	54.583	57.31215

3.4.5. Les Requêtes conditionnelles (Instructions CASE dans Oracle)

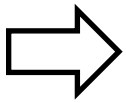
La clause SQL « CASE » est une expression conditionnelle générique, similaire aux instructions if/else dans d'autres langages de programmation. Elle permet d'accorder deux valeurs différentes à un attribut calculé en fonction d'une condition. Utilisée avec « SELECT », « WHERE », ….

Syntaxe :

```
CASE
    WHEN condition_1  THEN result_1
    WHEN condition_2  THEN result_2
    [WHEN ...]
    [ELSE else_result]
END
```

Exemple-1 : Soit la relation où on stock les moyennes des étudiants dans un module :
NoteModule(idetudiant, Moy)
Créer requête SQL permettant d'afficher les étudiants avec leur moyenne et un nouveau attribut où on calcule la décision : « Ajourné » si la Moy<5 , « Rattrapage » si la 5<=Moy<10 et « Validé » si la Moy>=10 :

```
select idetudiant, Moy,
case
when Moy<5 then 'Ajourné'
when Moy<10 then 'Rattrapage'
else 'Validé'
end  as decision
from NoteModule;
```



idetudiant [PK] integer	moy double precision	decision text
1	12.5	Validé
2	10	Validé
3	8	Rattrapage
4	4.5	Ajourné
5	15	Validé
6	3	Ajourné
7	5.5	Rattrapage
8	11.5	Validé
9	10.05	Validé
10	2.5	Ajourné

Exemple-2: Dans la BDD « Gestion des Ventes », afficher les commandes réalisées le mois dernier :

```
select idcommande, datecommande
from commandes
where extract(month from datecommande)=
case when extract(month from current_date)=1 then 12
else extract(month from current_date)-1
end
and
extract(year from datecommande)=
case when extract(month from current_date)=1 then extract(year from current_date)-1
else extract(year from current_date)
end;
```


3.4.6. Les Sous-Requêtes

Dans le langage SQL une sous-requête (aussi appelé requête imbriquée ou requête en cascade) consiste à exécuter une requête à l'intérieur d'une autre requête. Une requête imbriquée est souvent utilisée au sein d'une clause WHERE.

Syntaxe

Il y a plusieurs façons d'utiliser les sous-requêtes. De cette façon il y a plusieurs syntaxes envisageables pour utiliser des requêtes dans des requêtes.

a- Requête imbriquée qui retourne un seul résultat :

Dans l'exemple ci-dessous on veut afficher les produits dont le prix est supérieur au prix du produit id=10.ce dernier est trouvé par une sous requête.

```
select * from produits
```

where prixunitaire > **(select prixunitaire from produits where idproduit = 10)**;

b- Requête imbriquée qui retourne une colonne (La clause IN et NOT IN) :

Une requête imbriquée retourne dans ce cas une colonne entière. Dès lors, la requête externe peut utiliser la commande IN ou NOT IN pour filtrer les lignes qui possèdent une des valeurs retournées par la requête interne. La clause IN vérifie la concordance d'une à plusieurs données.

Syntaxe

```
SELECT *
FROM `table`
WHERE `nom_colonne` IN (
    SELECT `colonne`
    FROM `table2`
    WHERE …)
```

Exemple : Soit la relation suivante où on représente les villes visitées par personne.

idpersonne integer	nom text	ville text
1	Nom1	Casablanca
1	Nom1	Rabat
2	Nom2	Casablanca
2	Nom2	Rabat
2	Nom2	Fes
3	Nom3	Casablanca
3	Nom3	Rabat
3	Nom3	Fes
4	Nom4	Casablanca
2	Nom2	Rabat

On veut savoir les personnes qui n'ont pas visité « Rabat »

select distinct idpersonne, nom from personne where **idpersonne not in** (select **idpersonne** from **(select * from personne where ville = 'Rabat')** as R);

$$\pi_{idpersonne}(personne) - \pi_{idpersonne}(\sigma_{ville='Rabat'}(personne))$$

idpersonne integer	nom text
4	Nom4

Remarques :

- Pour éviter des redondances dans les résultats de SELECT il faut ajouter la clause **DISTINCT** après le mot SELECT.

c- Requête imbriquée qui retourne des tuples (La clause Exists et NOT exists):

Une requête imbriquée retourne dans ce cas un ensemble de tuples. Dès lors, la requête externe peut utiliser la commande EXISTS ou NOT EXISTS pour filtrer les lignes qui sont égales aux lignes retournées par la requête interne. La clause EXISTS s'utilise dans une clause conditionnelle pour savoir s'il y a une présence ou non de lignes lors de l'utilisation d'une sous-requête. Deux tuples de même structure sont les mêmes si :

- S'ils ont un identifiant ils doivent avoir les mêmes valeurs
- S'ils n'ont pas d'identifiants il faut que les valeurs de tous les attributs soient les mêmes. Cela se traduit par la condition suivante qui doit être intégrée dans la sous requête :
  `where R1.a=R2.a and R1.b=R2.b and R1.c=R2.c and …..`

Ici c'est les opérations algébriques *intersection* et *différence* qui vont être utilisé par le SGBDR.
Remarque : Les clauses IN et NOT IN peuvent aussi jouer le rôle des deux opérations *intersection* et *différence* si les valeurs dont on cherches la concordance appartiennent à un identifiants (clé primaire).

Syntaxe

```
SELECT nom_colonne1, nom_colonne2
FROM `table1`
WHERE EXISTS (
   SELECT nom_colonne1, nom_colonne2
   FROM `table2`
   WHERE nom_colonne1= table1. nom_colonne1
        AND nom_colonne2= table1. nom_colonne2)
```


Attention : cette commande n'est pas à confondre avec la clause IN. La commande EXISTS vérifie si la sous-requête retourne un résultat ou non, tandis que IN vérifie la concordance d'une à plusieurs données.

Exemple : On peut écrire la requête des personnes qui n'ont pas visité « Rabat » de la manière suivante :

```
select distinct idpersonne, nom from personne as p where not exists (select * from
personne where ville = 'Rabat' and p.idpersonne=idpersonne and p.nom=nom);
```

Noter : ici pour bien préciser les attributs des requêtes interne et externe on renomme les tables de la requête externe (as p).

3.4.7. Requête multi tables

Sont des requêtes où on utilise dans *la projection* et/ou dans *la sélection* des données provenant de plusieurs tables.

a- Produit cartésien

Le produit cartésien est une opération ensembliste commitative et associative portant sur deux relations R1 et R2 qui n'ont pas le même schéma et qui construit une troisième relation regroupant exclusivement toutes les possibilités de combinaison des tuples des relations de R1 et R2. Cela se traduit en SQL par

Select * From R1, R2	$R1 \times R2$

Cette opération est utilisée pour réaliser les jointures entre les tables de la base de données. Elles utilisée aussi pour faire d'autre opérations comme la différence et la division.

Exemple : Soit le MR suivant qui modélise les villes visitées par personne.

personnes(idpersonne, nom)
villes(idville, nomville)
visiteville(#idpersonne, #idville)

1- Quelles sont les villes non visitées par personne :

```
select * from personnes p, villes v
where not exists (select * from visiteville
where idpersonne=p.idpersonne and idville=v.idville);
```

En algèbre relationnelle on écrit :

$$R = \pi_{idpersonne,idville}(personnes \times villes) - visiteville$$
$$\pi_{idpersonne,nom,nomville}(personnes \bowtie R \bowtie villes)$$

2- Quelles sont les personnes qui ont visité toutes les villes

```
select * from personnes where idpersonne not in
(select idpersonne from (select * from personnes p, villes v
where not exists (select * from visiteville where idpersonne=p.idpersonne and
idville=v.idville)) as R);
```

En algèbre relationnelle on écrit :

$$R = \pi_{idpersonne}(personnes) - \pi_{idpersonne}(\pi_{idpersonne,idville}(personnes \times villes) - visiteville)$$
$$\pi_{idpersonne,nom}(personne \bowtie R)$$

b- La jointure

La **thêta jointure** est une sélection dans un produit cartésien. Dans le cas où cette sélection utilise

une égalité entre deux attributs des deux relations on parle de **l'équi-jointure**. Cette dernière devient une **jointure naturelle** si elle est réalisée entre deux attributs de mêmes types de données et même non d'attribut.

Exemple : dans le MR de gestion des ventes, si on veut afficher les commandes de chaque client on écrit en SQL :

```
Select Clients.idclient, nomclient, idcommande
From Clients, Commandes
Where Clients.idclient=Commandes.idclient
```

En AR :

$\pi_{idclient,nomclient,idcommande}(Clients \bowtie Commandes)$

Remarque : Dans cette requête, l'attribut « idclient » existe dans les deux tables « clients » et « commandes ». Pour éviter un ambigüité on précise à partir de quelle table on fait la projection « idclient » (**Clients.idclient)**.

c- Jointure interne (Inner Join)

La jointure interne ou l'equi-jouinture est utilisée pour retourner les tuples quand la condition d'égalité des attributs des deux tables est vraie. Cette commande retourne les enregistrements lorsqu'il y a au moins une ligne dans chaque colonne qui correspond à la condition.

Syntaxe :

```
Select Clients.idclient, nomclient, idcommande
From Clients Inner join Commandes
on Clients.idclient=Commandes.idclient
```

N.B : Dans le d'une jointure interne la requête n'affiche pas les clients qui n'ont pas de commandes.

Clients

idcl	Nom	Ville
1	Ahmed	Casa
2	Amine	Casa
3	Anas	casa

Commandes

idCom	Date	idcl
1	12/02/2021	1
2	15/03/2021	2
3	25/03/2021	1
4	02/02/2022	3
5	12/02/2022	3

Quelles sont les commandes (idcom , date) de chaque client ?

$\pi_{clients.idclient,Nom,idcom}(\text{Clients} \bowtie Commandes)$

idclient	Nom	idCom	Date
1	Ahmed	1	12/02/2021
1	Ahmed	3	25/03/2021
2	Amine	2	15/03/2021
3	Anas	4	02/02/2022
3	Anas	5	12/02/2022

d- Jointure externe (Left Join ou Right Join)

Dans la sélection précédente, un client qui n'a pas réalisé de commande n'apparaîtra jamais dans la liste, puisqu'il n'y aura dans le produit cartésien des deux tables aucun élément où l'on trouve une égalité des colonnes idclient.
On pourrait pourtant désirer afficher une liste de tous les clients, avec leurs commandes, sans omettre les clients sans commandes. On écrira alors :

Syntaxe :

```
Select Clients.idclient, nomclient, idcommande
From Clients Left join Commandes
on Clients.idclient=Commandes.idclient
```

La jointure externe ajoute des lignes fictives dans une des tables pour faire la correspondance avec les lignes de l'autre table. Dans l'exemple précédent, une ligne fictive (une commande fictive) est ajoutée dans la table des commandes si un client n'a pas de commande. Cette ligne aura tous ses attributs NULL, sauf celui des colonnes de jointure.
LEFT indique que la table dans laquelle on veut afficher toutes les lignes (la table clients) est à gauche de LEFT OUTER JOIN. C'est dans l'autre table (celle de droite) dans laquelle on ajoute des lignes fictives. De même, il existe RIGHT OUTER JOIN qui est utilisé si on veut afficher toutes les lignes de la table de droite (avant le RIGHT OUTER JOIN) et FULL OUTER JOIN si on veut afficher toutes les lignes des deux tables.

Clients

idcl	Nom	Ville
1	Ahmed	Casa
2	Amine	Casa
3	Anas	Casa
4	Ali	Rabat

Commandes

idCom	Date	idcl
1	12/02/2021	1
2	15/03/2021	2
3	25/03/2021	1
4	02/02/2022	3
5	12/02/2022	3

Quelles sont les commandes (idcom , date) de chaque clients ? Afficher tous les clients

$\pi_{clients.idclient,Nom,idcom}(\text{Clients} \bowtie Commandes)$

idclient	Nom	idCom	Date
1	Ahmed	1	12/02/2021
1	Ahmed	3	25/03/2021
2	Amine	2	15/03/2021
3	Anas	4	02/02/2022
3	Anas	5	12/02/2022
4	**Ali**	**Null**	**Null**

3.4.8. Supprimer les doubles avec DISTINCT

Supposons que nous voulions la liste des clients ayant acheté quelque chose. Nous voulons que chaque client ayant acheté quelque chose ne soit affiché qu'une seule fois – nous ne voulons pas savoir ce qu'a acheté chaque client – nous voulons juste connaître les clients qui ont acheté quelque chose. Pour cela, nous allons devoir supprimer les doubles du résultat de la sélection pour n'afficher les clients qu'une seule fois. Pour cela, nous allons utiliser la clause DISTINCT.
Nous allons d'abord faire une semi-jointure entre les tables Clients et Commande, et ajouter la clause DISTINCT sur le champ ou la répétition peut se produire :

```
Select clients.idclient, nomclient
From Clients inner join Commandes
on Clients.idclient=Commandes.idclient;
```

Si on exécute cette requête directement, SQL va nous renvoyer une liste des numéros, et noms correspondants aux noms et prénoms des clients ayant passé chaque commande, il est clair qu'un client ayant passé plusieurs commandes va se retrouver plusieurs fois dans cette liste.

```
Select distinct clients.idclient, nomclient
From Clients inner join Commandes
on Clients.idclient=Commandes.idclient;
```

En indiquant la clause DISTINCT avant le champ idclient, on indique à SQL qu'on ne veut pas voir apparaître plusieurs fois un client ayant ce numéro dans la sélection renvoyée.
On peut même rendre le résultat de la sélection plus agréable à la lecture en utilisant la clause ORDER BY :

3.4.9. Les Requêtes de synthèses avec agrégations et regroupement.

L'SQL nous permet de faire des synthèses sur des attributs en regroupant un ensemble de tuples. Pour cela des fonctions d'agrégation sont utilisées.

a- Les fonctions d'*agrégats*

SQL a cinq fonctions importantes : SUM, AVG, MAX, MIN et COUNT. On les appelle fonctions d'ensemble par ce qu'elles synthétisent le résultat d'une requête plutôt que de renvoyer une liste d'enregistrements.

Fonction	Signification
SUM ()	Donne le total d'un champ de tous les enregistrements satisfaisant la condition de la requête. Le champ doit bien sur être de type numérique
AVG()	donne la moyenne d'un champ de tous les enregistrements satisfaisant la condition de la requête
MAX()	donne la valeur la plus élevée d'un champ de tous les enregistrements satisfaisant la condition de la requête
MIN()	Donne la valeur la plus petite d'un champ de tous les enregistrements satisfaisant la condition de la requête.
COUNT()	Renvoie le nombre d'enregistrements satisfaisant la requête.

i- Utilisation simple :

L'utilisation la plus générale consiste à utiliser la syntaxe suivante :

```
SELECT COUNT(*) FROM table
```

Pour compter le nombre total de ligne d'une table, L'utiliser l'étoile "*" signifie que l'on cherche à compter le nombre d'enregistrement sur toutes les colonnes. C'est le degré d'agrégation le plus élevé. La syntaxe serait alors la suivante :

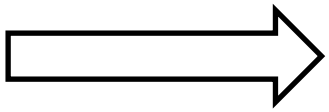
ii- Utilisation avec GROUP BY

L'utilisation de ces fonctions avec la commande GROUP BY permet de diminuer le degré d'agrégation en construisant des sous-ensembles de tuples sur lesquels on applique la synthèse :

Exemples1 : Nous voulons connaitre le nombre de clients par ville. Pour cela il faut regrouper les villes et compter les clients.

```
select ville, count(idclient) as nbre from clients
group by ville;
```

ville text	nbre integer
Sanliurfa	831
Sandviken	568
San Vicente	186
San Ramón	649
San Ramón	957
San Rafael	27
San Rafael	121
San Rafael	227
San Rafael	674
San Rafael	940
San Pedro	122
San Pedro	782
San Nicolás	937



ville text	nbre bigint
Sanliurfa	1
Sandviken	1
San Vicente	1
San Ramón	2
San Rafael	5
San Pedro	2
San Nicolás	1

Exemple2: Calculer le montant total des ventes par année. Pour faire cette synthèse on aura besoin de la « datecommande », « prixunitaire », et la « quantite ». Ces données provinnent des tables « Commandes », « produits », et « lignecommandes ». Par conséquent deux jointure doivent être utilisées par la requête.

```
select extract(year from datecommande) as ANN,
SUM(Prixunitaire*Quantite) as Mt
From commandes, lignecommandes, produits
where commandes.idcommande=lignecommandes.idcommande
and lignecommandes.idproduit=produits.idproduit
group by extract(year from datecommande);
```

ann double precision	mt double precision
2012	23644571.85
2011	23013603.065
2014	22337362.333
2013	22852960.509
2019	22333677.811
2020	19043467.43
2016	22094122.45299
2010	3888798.659
2017	22617926.482
2015	22902964.15
2018	21741646.19899

Exemple3 : Si on introduit dans la projection la ville on va analyser le montant total des ventes par année et par ville. Puisque la « ville » est dans la table « clients » une 3ème jointure doit être utilisée.

```
select extract(year from datecommande) as ANN, ville,
SUM(Prixunitaire*Quantite) as Mt
```

From clients, commandes, lignecommandes, produits
where clients.idclient=commandes.idclient
and commandes.idcommande=lignecommandes.idcommande
and lignecommandes.idproduit=produits.idproduit
group by extract(year from datecommande), ville;

ann double precision	ville text	mt double precision
2010	Aguacaliente	22005.967
2010	Anantapur	19242.37
2013	Whangarei	41382.836
2013	Whitchurch	8861.688
2013	Whyalla	70653.618
2013	Wilmont	134647.555
2013	Woerden	24237.333
2013	Woodstock	40854.083
2013	Zaltbommel	64984.752
2013	Zaria	110915.465
2013	Zoetermeer	50790.491
2014	Abeokuta	33420.13
2014	Adana	22321.859
2014	Aguacaliente	36248.656
2014	Åkersberga	17821.635

iii- Utilisation de la clause Having

La clause « **Having** » remplace la clause « **Where** » dans le cas d'une Sélection sur une relation résultante d'une agrégation. c-a-d après Group by on ne peut pas utiliser la clause Where.
exemple : On veut savoir le chiffre d'affaire réalisé pendant l'année 2021. il ya deux manière pour écrire cette requête :

1- Première manière c'est de demander au système de filtrer les commandes de 2021 puis applique la fonction d'agrégat sum() sur les tuples de la relation résultante de la jointure entre « commandes de 2021 », « lignecommandes » et « produits ».

```
select SUM(Prixunitaire*Quantite) as Mt
From commandes, lignecommandes, produits
where commandes.idcommande=lignecommandes.idcommande
and lignecommandes.idproduit=produits.idproduit
and extract(year from datecommande)=2022;
```

2- Première manière c'est de demander au système d'appliquer de regrouper les années puis appliquer la fonction d'agrégat sum() sur les tuples de chaque groupe d'année et après filtrer l'année 2021.


```
select SUM(Prixunitaire*Quantite) as Mt
From commandes, lignecommandes, produits
where commandes.idcommande=lignecommandes.idcommande
and lignecommandes.idproduit=produits.idproduit
group by extract(year from datecommande)
having extract(year from datecommande)=2022;
```

l'optimiseur du SGBD choisi la manière-1 puis c'est elle la plus optimale.